AIM/Trend Add-In
USER'S MANUAL

October 2002

AIM Project Team

Table of Contents

1. What is AIM/Trend Add-In? 1
2. Installation and Uninstallation 2
   2.1. Installation 2
   2.2. Uninstallation 2
      2.2.1 Temporary uninstall 2
      2.2.2 Complete uninstall 3
3. Basic Operations 4
   3.1. Toolbar commands 4
   3.2. Executing programs 4
   3.3. Executing macros 4
   3.4. Saving data in a worksheet as a text file 4
   3.5. Loading data from a text file into a worksheet 5
4. Macro Syntax 6
   4.1. What is ATML? 6
   4.2. Format 6
   4.3. Command statements 6
5. Program Syntax 9
   5.1. What is ATPL? 9
   5.2. Format 9
   5.3. Variables, Expressions, Characters, and Indexes 10
      5.3.1 Variable types 10
      5.3.2 Variable names 10
      5.3.3 Expressions 10
      5.3.4 Time-series variables 11
      5.3.5 Index function 12
      5.3.6 Substituting characters during execution 14
   5.4. Declarative statements and Data statements 14
   5.5. I/O statements 20
      5.5.1 Auxiliary input/output statements 20
      5.5.2 Standard output 22
      5.5.3 Input statements 22
      5.5.4 Output statements 28
      5.5.5 Handling database statements 33
   5.6. Assignment statements 39

5.7. Regression calculation and interpolation 43
5.8. Control statements 46
5.9. Executing subprograms and VBA macros 48
5.10. Handling files and directories 49
5.11. Handling worksheets 51
5.12. Handling workbooks 53
5.13. Other statements 55
6. Creating User Interfaces 57
7. Index of ATPL Instructions 59

1. What is AIM/Trend Add-In?

AIM/Trend Add-In is provided as an add-in file for Microsoft Excel 2000 (or later version) and is used the file name 'AimTrend.xla'. This add-in file is designed for personal computers running Microsoft Windows 95 (or later version) and with Excel 2000 (or later version) installed.

AIM/Trend Add-In supports a simple programming language known as ATPL (AIM/Trend Program Language). ATPL is a language system optimized for manipulating time-series variables. For example, it can be used to write a simple program that reads time-series data from a worksheet, performs regression calculation, then comes up with a future estimation and outputs its results to a worksheet.

AIM/Trend Add-In also supports a macro language called ATML (AIM/Trend Macro Language). Using this macro language, processes such as loading and saving text files, executing ATPL programs, and printing worksheets can be performed as one series of actions.

With a basic knowledge of Microsoft Visual Basic for Applications (VBA), simple user interfaces can be created by adding ATML macros as user form buttons.

2. Installation and Uninstallation

2.1. Installation

(1) Installing the MS Query add-in to Excel

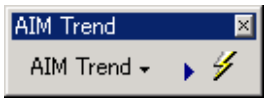
Install the MS Query add-in to Excel by following the steps below. If this is the first time you are installing the MS Query add-in to Excel, the Excel application CD-ROM may be required. The MS Query add-in must be installed before installing the AIM/Trend add-in to Excel.

- Start Excel.
- Select [Tools]/[Add-Ins] from the menu bar to display the [Add-Ins] dialog box.
- Select the MS Query Add-in, which is listed in the dialog box, and click [OK].

(2) Installing the AIM/Trend Add-In to Excel

Install the AIM/Trend add-in to Excel by following the steps below.

- Start Excel.
- Select [Tools]/[Add-Ins] from the menu bar to display the [Add-Ins] dialog box.
- Click the [Browse...] button. In the file selection window, select the file 'AimTrend.xla' from the AIM/Trend CD-ROM and click [OK].
- A confirmation window will appear asking if it's OK to copy the 'AimTrend.xla' file. Click [OK] in this window.
- Make sure that “AIM/Trend Add-In" is now showing in the [Add-Ins] dialog box. Click [OK]. The [AIM/Trend] toolbar will now appear. The AIM/Trend add-in is now successfully installed in Excel.



📖 The [AIM/Trend] toolbar can be shown or hidden by selecting or deselecting it on the [View]/[Toolbars] submenu from the Excel menu bar.

2.2. Uninstallation

2.2.1 Temporary uninstall

To uninstall the AIM/Trend Add-In, follow the steps below.

- Start Excel, and select [Tools]/[Add-Ins] from the menu bar to display the [Add-Ins] dialog box.
- Deselect "AIM/Trend Add-In" from the list box, and click [OK]. This completes the

uninstallation of the AIM/Trend add-in.

🕮 Generally speaking, the more add-ins that are installed in Excel, the longer it takes for Excel to start up. If this is not convenient, add-ins that are not used often can be temporarily uninstalled following the steps described above. Once an Excel add-in is installed, it can be installed or uninstalled (enabled or disabled) simply by selecting or deselecting it from the [Add-Ins] dialog box.

2.2.2 Complete uninstall

Add-ins in Excel do not disappear from the list in the [Add-Ins] dialog box merely by "temporarily uninstalling" them. If you prefer to delete an add-in from the list, follow the steps below.

- Find and note the location of the file 'AimTrend.xla' by using the [Version Info...] command on the toolbar.
- Follow the steps described in "Temporary uninstall" above to disable the AIM/Trend add-in.
- Using Windows Explorer or other means, delete file 'AimTrend.xla', which is located in the folder noted above.
- Start Excel and select [Tools]/[Add-Ins] from the menu bar to display the [Add-Ins] dialog box.
- Select the check box for "AIM/Trend Add-In".
- The message "Cannot find add-in 'AimTrend.xla', Delete form list?" will appear. Now click [OK] to delete "AIM/Trend Add-In" from the list.

3. Basic Operations

3.1. Toolbar commands

The [AIM/Trend] toolbar includes a drop-down menu with the following commands.

Command	Description
Execute Program	Executes programs written in ATPL.
Execute Macro	Executes macros written in ATML.
Save Data...	Saves data in a worksheet as a text file.
Load Data...	Loads data from a text file into a worksheet.
Version Info...	Displays version information for the AIM/Trend add-in.

3.2. Executing programs

Programs can be executed by following the steps below. For more on how to write programs, refer to "5. Program Syntax".

- Activate the worksheet containing the program to be executed.
- Move the cursor so that it is inside the program to be executed.
- Click the [AIM/Trend] button on the [AIM/Trend] toolbar, and select [Execute Program] from the drop-down menu to start the program.

3.3. Executing macros

Macros can be executed by following the steps below. For more on how to write macros, refer to "4. Macro Syntax".

- Activate the worksheet containing the macro to be executed.
- Move the cursor to the first row of the macro.
- Click the [AIM/Trend] button on the [AIM/Trend] toolbar, and select [Execute Macro] from the drop-down menu to start the macro. The rows to be executed will be those extending from the row on which the cursor is placed to the row just before the first row with a blank column B.
- After the macro has executed, the message "Normal end" will be displayed.

3.4. Saving data in a worksheet as a text file

Data in a worksheet can be saved as a tab delimited text file by using the [Save Data...] command on the drop-down menu on the [AIM/Trend] toolbar. Data in worksheets are not saved as 'values', but as 'formulas'. To save data, follow the steps below.

- Activate the worksheet containing the data to be saved.

- Click the [AIM/Trend] button on the [AIM/Trend] toolbar, and select [Save Data...] from the drop-down menu. A dialog box for saving the data will appear.
- Type the name of the file into the dialog box and click [Save].
- The data in the worksheet will now be saved as a tab delimited text file.

3.5. Loading data from a text file into a worksheet

The contents of a text file (tab delimited) can be loaded into a worksheet by selecting the [Load Data...] command from the drop-down menu on the [AIM/Trend] toolbar. To load text file data into a worksheet, follow the steps below.

- Activate the worksheet into which data will be read.
- Click the [AIM/Trend] button on the [AIM/Trend] toolbar, and select [Load Data...] from the drop-down menu. A dialog box for loading data will appear.
- Type the name of the file into the dialog box and click [Load]. The data in the specified text file will now be loaded into the worksheet.

4. Macro Syntax

4.1. What is ATML?

ATML (AIM/Trend Macro Language) is a macro programming language used for executing a series of processes such as loading and saving text files, running ATPL programs, and printing worksheets.

4.2. Format

The basic format of a macro is described below.

- Not case sensitive.
- Each statement consists of a command and one or more argument(s). Each statement is always written on a separate row. In a statement row, the command is written in column B, and arguments are written in columns C to E.
- Rows containing an asterisk (*) in column A are treated as comment rows.
- The rows to be executed will be those extending from the row on which the cursor is placed to the row just before the first row with a blank column B. (For more on how to execute macros, see "3.3 Executing macros".)

4.3. Command statements

➢ **Run statement**

Used to execute an ATPL program. If the program specified by the Run statement does not exist, an error message is displayed and the process is canceled.

[Format]

A	B	C	D	E
	Run	*SheetName*	*ProgramName*	*Argument; Argument; ...*

[Arguments]

Argument	Description
SheetName	Name of worksheet containing the program to be executed
ProgramName	Name of program to be executed
Argument	Argument that is passed to the ATPL program. If more than one argument is used, each is delimited by a semicolon (;).

➢ **VBA statement**

Used to execute a VBA macro. This statement cannot execute a macro that contains arguments.

[Format]

A	B	C	D	E
	VBA	*MacroName*		

[Arguments]

Argument	Description
MacroName	Name of VBA macro to be executed

- **Load and Load- statement**

Used to load data from a text file into a worksheet. Tabs are used as delimiters in the text. If the file specified by the Load statement does not exist, an error message is displayed and the process is canceled. In such cases, the Load statement does not do anything.

[Format]

A	B	C	D	E
	Load	*SheetName*	*DirectoryName*	*FileName*

[Arguments]

Argument	Description
SheetName	Name of worksheet
DirectoryName	Name of directory where the file is located
FileName	Name of the file containing the data

- **Save statement**

Used to save data in a worksheet as a text file. If the worksheet specified by the Save statement does not exist, an error message is displayed and the process is canceled.

[Format]

A	B	C	D	E
	Save	*SheetName*	*DirectoryName*	*FileName*

[Arguments]

Argument	Description
SheetName	Name of worksheet
DirectoryName	Name of directory
FileName	Name of file

- **DelWs statement**

Used to delete a worksheet. If the specified worksheet does not exist, no process will be performed.

[Format]

A	B	C	D	E
	DelWs	*SheetName*		

[Arguments]

Argument	Description
SheetName	Name of worksheet to be deleted

- **PrintWs statement**

Used to print the contents of a worksheet. If the specified worksheet does not exist, an error message is displayed and the process is canceled. Printing is performed using the currently set printer.

[Format]

A	B	C	D	E
	PrintWs	*SheetName*		

[Arguments]

Argument	Description
SheetName	Name of worksheet to be printed

5. Program Syntax

Symbols used in the following descriptions:

- Arguments in column C and those whose names begin with an exclamation mark (!) are called labels. In statement format descriptions, arguments other than labels are shown in italics so they can be easily distinguished from labels.
- In statement format descriptions, an argument within parentheses ( ) indicates that it may be omitted.
- In statement format descriptions, a label within parentheses ( ) indicates that the rows containing that label may be omitted. Omitting labels means that arguments in columns D and F for that row are also omitted.
- In statement format descriptions, an argument within angle brackets < > indicates that data type is character string and that it is specified by using expression.
- In statement format descriptions, a row that ends with the ellipsis symbol (...) indicates that the row above that row may be repeated.

5.1. What is ATPL?

AIM/Trend Program Language (ATPL) is a simple programming language designed to handle time-series data in a simple way. The AIM/Trend software includes an interpreter that executes programs written in ATPL.

5.2. Format

The interpreter does not distinguish between uppercase and lowercase letters.

For the interpreter to recognize the body of a program, the first row in the program must contain the word **Program** in column B, and the last row in the program must contain the word **End** in column B.

Programs consist of one or more **statements**, while statements consist of a **command** and one or more **arguments**. The command is written in column B, and the arguments are written in columns C to F. One statement consists of one or more rows. For more on individual statement formats, see the descriptions for the respective statements.

A row that begins with an asterisk (*) in either column A or B is a **comment statement**. For statements consisting of multiple rows, the entire statement becomes a comment statement if the first row is defined as a comment row. When a program is executed, the color of the comment statements changes to green.

It is possible to split an argument in column D into more than one row by using the ampersand (&) symbol in column C. There are no functions defined for arguments contained in columns other than

column D.

The ATPL interpreter ignores the colon (:) symbol when used as the first character of an argument. This enables the interpreter to avoid input numerals to automatically shift to the right edge.

A	B	C	D	E	F
	Program				
	command	*argument*	*argument*	*argument*	*argument*
		argument	*argument*	*argument*	*argument*
		&	*argument*		
*	*command*	*argument*	*argument*		
	**command*	*argument*	*argument*		
	End				

5.3. Variables, Expressions, Characters, and Indexes

5.3.1 Variable types

In ATPL, variables are sorted into two types: time-series types and non-time-series types. Variables sorted into time-series types are called **time-series variables**, while variables sorted into non-time-series types are called **parameters**.

ATPL variables can be handled without having to distinguish between characters and values. (Internally, variables are processed as a "variant type" variable of VBA.)

5.3.2 Variable names

The rules below describe the naming of variables.

- No limit on how many characters can be used.
- Not case sensitive.
- Alphanumeric characters and four special characters, underscore (_), colon (:), left bracket ([), and right bracket (]) can be used in variable names. (Any character except the underscore (_) may be used in array descriptions.)
- Variable names consisting only of numerals (digits) are not permitted.
- The names "C", "R" and "RC" are not permitted.

5.3.3 Expressions

A series of variables and parameters connected by parentheses and worksheet operators is called an expression. Character constants are placed in double quotation marks. Excel worksheet functions can also be used in expressions. However, a worksheet function that requires a cell range or array as an argument cannot be used.

5.3.4 Time-series variables

The time-series variable in ATPL is a unique concept not used in other programming languages. Below, the concept and use of time-series variables are explained using an example program.
The example program below calculates the GDP per capita from the population and GDP.

Example program **SeriesTest**

A	B	C	D
	Program	SeriesTest	
	Series	YEAR	DIR;1990,1995,2000
	Data	POP	1000,1010,1020
	Data	GDP	1000,1050,1100
	CalSrs	GDPCAP	GDP/POP
		IDX	GDPCAP/GDPCAP{1990}*100
*		IDX	GDPCAP/GDPCAP{f}*100
	End		

(1) What are time-series variables?

Time-series variables are arrays that contain within them a dimension of time. An array element of time dimension is known as a time element. Whenever using time-series variables, the Series statement must be used before them. The role of the Series statement is to declare the number and names of time elements to be used. In the above example, 3 time elements, 1990, 1995, and 2000, are declared. Any time-series variables declared after this point will all have the three time elements.

Executing the Series statement also automatically declares time-series variables containing time element names. These time-series variables are also called time variables. The names of the time variables (also known as the "time series names") will be the character string contained in column C. In this example, 'YEAR' is used as the name of the time variable.

For details on how to use the Series statement, see "5.4. Declarative statements and Data statements".

(2) Setting initial values

The **Data statement** is used to declare time-series variables and set their initial values. In the example above, two Data statements are used to declare the time-series variables POP and GDP, and then to assign initial values to them. For details on how to use the Data statement, see "5.4. Declarative statements and Data statements".

(3) Assignments

To assign values to time-series variables, the **CalSrs statement** is used. The argument in column C

corresponds to the left side of the assignment expression, and the arguments in column D to the right side. The CalSrs statement is also used to declare time-series variables. When an undeclared time-series variable is used in column C, the variable is declared before any assignment occurs. In this example, GDPCAP (GDP per capita) is calculated from POP and GDP from the first row of the CalSrs statement. Because GDPCAP has not yet been declared by any preceding statements, it is declared by this statement. The assignments made by this statement can be expressed in mathematical form as follows:

GDPCAP(ti) = GDP(ti) / POP(ti) for all ti, ti = {1990, 1995, 2000}

For details on how to use the CalSrs statement, see "5.6. Assignment statements".

(4) Suffixes

By attaching a suffix surrounded by curly braces, { and }, a specific time element can be expressed. For example, GDPCAP{1990} means the GDP per capita in 1990. The second row of the CalSrs statement calculates variable IDX. IDX is the ratio of GDPCAP compared to the GDPCAP for 1990, which is set to 100. In addition, some suffixes have special meanings:

Suffixes	Description
{f}	The first time element
{l}	The last time element
{-}	The immediately previous time element
{+}	The immediately next time element

By using the {f} suffix, the expression to calculate IDX in the example can be expressed as:

GDPCAP/GDPCAP{f}*100

5.3.5 Index function

ATPL has a convenient function called the index function. In the example below, the index function is described by using an example program IndexTest.

Example program **IndexTest**

	A	B	C	D	E	F
1		Program	IndexTest			
2		Index	$i	Coal,Oil,Gas		
3		CalCns	A[Coal]	:1		
4		CalCns	A[Oil]	:3		
5		CalCns	A[Gas]	:2		
6		CalCns	A[$i]	A[$i]*1000		#

7		CalCns	TOT	sum($i,A[$i])		#
8		CalCns	A[$i]	A[$i]/TOT		#
9		End				

To use the index function, an **Index statement** must first be executed to declare the index and its members. The Index statement in the second row above declares an index named $i containing three members, Coal, Oil, and Gas. Mathematically, it is written as follows:

i = {Coal, Oil, Gas}

The first character of an index name is always the dollar sign ($). A maximum of 50 indexes can be declared in a program. For details on how to use the Index statement, see "5.4. Declarative statements and Data statements".

The CalCns statement from rows 3 to 5 declares three parameters, A[Coal], A[Oil], and A[Gas], and then assigns values to these parameters. Be aware that the form "A[ ]" does not mean that it is an array, but instead that it contains characters [and] in the name of parameter itself. (Four special characters _ [ ] : can be used in variable names.) **There is no concept of an array in ATPL.**

In the CalCns statement in the 6th row, index $i is used. The interpreter looks at this row as follows.

$A_i = A_i * 1000$ for all i, i={Col, Ool, Gas}

In this instance, the interpreter does not recognize and process A[$i] as "three elements of array A," but does recognize and process it as "three different parameters named A[Coal], A[Oil], and A[Gas]."

Statements that contain an index only on the right side as follows will result in errors.

B	C	D
Calcns	B	A[$i]

Indexes are used in statements that use variables as arguments in column C or D. Indexes are recognized by the interpreter in the following cases.

- When it is used by itself.
- When the indexes are separated by one of the nine special characters, namely

  ( ) [ ] : ; , . &

Indexes written in other forms are not recognized as indexes by the interpreter.

Recognizable example	$i	TPS[$i]	($i)	TPS.$i
Unrecognizable example	TPS$i	TPS-$i	TPS_$i	

Also, because the colon symbol (:) can be used in variable names or used as a delimiter for an index, it is possible to use this symbol to insert more than one index into one variable and process the variable, much like the case of a multi-dimension array.

B	C	D
CalCns	D[$i:$j:$k]	x[$i]*y[$j]*z[$k]

The Index function also includes what is called the SUM expression. The SUM expression can be used to easily perform calculation of a sum. In the CalCns statement of the 7th row the following calculations are performed:

$TOT = \sum_i A_i \qquad i=\{Coal, Oil, Gas\}$

However, it is not possible to insert a SUM expression into another SUM expression.

5.3.6 Substituting characters during execution

The ATPL interpreter decomposes the character strings in column C or D in a program by using twelve special characters

( ) [ ] : ; , . & - \

as delimiters, and replaces character strings that have the percent (%) symbol as the first character by parameter values that are the same character strings except without the % symbols.

For example, when values of parameters A and B are respectively assigned as "Coal" and 1990, then TPS.%A, TPS[%A] and GDP{%B} are respectively interpreted as TPS.Coal, TPS[Coal] and GDP{1990}.

5.4. Declarative statements and Data statements

- **Series and Series+ statements**

These statements are used to declare time elements (number and names of time elements) of time-series variables. For more on the role of the Series statement, see "5.3.4 Time-series variables". When the Series statement is executed again in a program, time-series variables that have already been declared are all cleared.

Series+ statements can be used to modify time elements that have already been declared earlier in the program. Series+ statements can be executed only when the Series statement has been executed earlier in the program. Execution of the Series+ statement causes the time elements that have already been declared and newly declared to be compared, and time elements with the same names will keep their old values. The formats of both the Series+ and Series statements are the same.

There are four formats to declare time elements. These formats can be used by combining in a single series statement.

(a) Writing time elements directly in the program

[Format]

B	C	D	E
Series	*TimeSeriesName*	!Dir;*Element*,*Element*,...	

[Arguments]

Arguments	Description
TimeSeriesName	Time-series name (time-variable name)
Element	Time element name

(b) Specifying time elements by start time, end time, and time interval.

[Format]

B	C	D	E
Series	*TimeSeriesName*	!Lin; *StartTime* * *EndTime* (, *Interval*)	

[Arguments]

Arguments	Description
TimeSeriesName	Time-series name (time-variable name)
StartTime	Start time
EndTime	End time
Interval	Time interval. This argument may be omitted. Default value is 1.

(c) Reading time elements from Time Series Type Data Table in a worksheet

[Format]

B	C	D	E
Series	*TimeSeriesName*	!Tab; *TableName*, *ItemName*	*SheetCode*

[Arguments]

Argument	Description
TimeSeriesName	Time-series name (time-variable name)
TableName	Name of Time Series Type Data Table* in the worksheet.
ItemName	Item name inside the Time Series Type Data Table*.
SheetCode	Sheet code of the worksheet that is set to the input device.

* For more on Time Series Type Data Tables, see the description of the ImpSrs statement.

(d) Reading time elements from Index Table in a worksheet

[Format]

B	C	D	E
Series	*TimeSeriesName*	!Idx; *TableName*	*SheetCode*

[Arguments]

Argument	Description
TimeSeriesName	Time-series name (time-variable name)
TableName	Name of Index Table* in the worksheet.
SheetCode	Sheet code of the worksheet that is set to the input device.

* For more on Index Tables, see the description of the Index statement.

[Example]

In the following example, eleven time elements, years 1990, 1991, ..., and 2000, are declared.

B	C	D	E	F
OpenWs	!import	Prd	<1	
series	year	!dir;1990,1991,1992		
		!lin;1993*1995,1		
		!tab;year,MyTab	<1	
		!idx;year	<1	

Contents of worksheet 'Prd'

A	B	C	D	E
#Table	MyTab			
year	1996	1997	1998	
	#Index	year		
		:1999		
		:2000		

- **Data statement**

The Data statement is used to declare time-series variables and set initial values for them.

[Format]

B	C	D
Data	*SerName*	*Value, Value, Value*

[Arguments]

Argument	Description
SerName	Name of time-series variable
Value	Initial value

[Example]

In the following example, a time-series variable POP is declared, and the initial values of time elements, for the years 1990, 1995, and 2000, are set to 1000, 1010, and 1020, respectively.

B	C	D
Series	YEAR	DIR;1990,1995,2000
Data	POP	1000,1010,1020

- **Dim statement**

The Dim statement is used to declare variables. Variables declared by this statement have a blank as their initial value.

[Format]

B	C	D
Dim	*VarType*	*VarName* (; *VarName; ...*)

[Arguments]

Argument	Description
VarType	Time-series variables: !Srs Parameters: !Cns
VarName	Names of variables. When specifying more than one, use a semicolon (;) as the delimiter.

[Example]

In the following example, parameters C1, C2, and C3, and time-series variables GDP and POP are all declared. Variables declared by this statement have a blank as their initial value.

B	C	D
Dim	!Cns	C1; C2; C3
Dim	!Srs	GDP; POP

➢ **Index statement**

The Index statement is used to declare indexes and their members. The first character of an index name must always be the dollar sign ($). Up to **50 indexes** and **100 members** can be declared in a program. There are three formats to declare indexes. These formats can be used by combining in one Index statement.

(a) Writing members directly in the program

[Format]

B	C	D
Index	*IndexName*	(!Dir;)* *MemberName, MemberName, ...*

* When the label is ommited, !Dir is specified.

[Arguments]

Argument	Description
IndexName	Name of index
MemberName	Name of member

[Example]

In the following example, $fossil is declared as indexes. Members of $fossil are "Coal", "Oil" and "Gas".

B	C	D	E	F
Index	$fossil	Coal,Oil,Gas		

(b) Reading members from Data Table in a worksheet

[Format]

B	C	D	E
Index	*IndexName*	!Tab; *TableName*	*SheetCode*

[Arguments]

Argument	Description
IndexName	Name of index
TableName	Name of Data Table in the worksheet
SheetCode	Sheet-code of the worksheet in which Data Table exists.

[Example]

In the following example, $fossil is declared as index. Members of $fossil are "Coal", "Oil" and "Gas".

B	C	D	E	F
OpenWs	!import	MySheet	<1	
Index	$fossil	!Tab;MyTab	<1	

Contents of worksheet 'Mysheet'

A	B	C	D
#Table	MyTab		
product	1990	1992	1993
Coal	10	11	12
Oil	20	21	22
Gas	30	31	32

(c) Reading members from Index Table in a worksheet

[Format]

B	C	D	E
Index	*IndexName*	!Idx; *TableName*	*SheetCode*

[Arguments]

Argument	Description
IndexName	Name of index
TableName	Name of Index Table in the worksheet
SheetCode	Sheet-code of the worksheet in which Index Table exists.

[Example]

In the following example, $fossil and $Sector are declared as indexes. Members of $fossil are "Coal", "Oil" and "Gas", and member of $Sector are "ARG", "IND" and "SRV".

B	C	D	E
OpenWs	!Import	MySheet	<1
Index	$Fossil	!Idx;Products	<1
	$Sector	!Idx;Sectors	<1

Index table written in worksheet 'MySheet'

A	B	C	D
	#Index	Product	
		Coal	
		Oil	
		Gas	
	#Index	Sector	
		AGR	
		IND	
		SRV	

First row can be any row.
Write '#INDEX' in column B of the first row.
Write the index name in column C of the first row.

When writing more than one index, always skip one row or more in between indexes.

- **ErsIdx statement**

The ErsIdx statement is used to erase index.

[Format]

B	C	D
ErsIdx	*IndexName*	

[Arguments]

Arguments	Description
IndexName	Name of Index to be erase

5.5. I/O statements

The most frequently used input/output device by ATPL is the Excel worksheet. By using simple instruction statements, ATPL can store worksheet data in the form of variables, as well as output data stored as variables into a worksheet.

ATPL is also able to use Microsoft Access databases file as input devices. For details, see "5.5.5 Handling database statements".

5.5.1 Auxiliary input/output statements

- **OpenWs statement**

When using worksheets for input or output, the worksheet first needs to be specified with an OpenWs statement. Specifying a worksheet as an I/O device is also referred to as "opening the worksheet." **Up to 20 worksheets** can be opened at the same time for both input and output.

Worksheets to be opened are assigned sheet codes. Sheet codes are used in input/output statements to identify the input source or output destination.

[Format]

B	C	D	E
OpenWs	*IOType*	*SheetName*	*SheetCode*
	…	…	…

[Arguments]

Argument	Description
IOType	Input/Output type For input: !Import For output: !Export, !Export+, !Export-
SheetName	Name of worksheet to be opened.
SheetCode	Sheet code assigned to the worksheet For input: Character string starting with less than sign (<). For output: Character string starting with greater than sign (>).

[Settings for opening]

When opening worksheet(s) for output, an **Output Row Counter** is set to each of the sheet codes. This counter is used in the management of output starting row.

	Process in case the specified sheet does not exist	Data clear	Setting of output starting row number.
!Import	Error	No	-
!Export	Adding sheet	Yes	1
!Export-	Adding sheet	No	1
!Export+	Adding sheet	No	*

* The row number of the last output row +1

[Example]

B	C	D	E
OpenWs	!Import	Book1.xls!WS_A	<1
	!Export	WS_B	>1

- **CloseWs statement**

The CloseWs statement is used to close all worksheets that are opening for input or output.

[Format]

B	C
CloseWs	

- **ShiftRow statement**

The ShiftRow statement is used to shift the Output Row Counter to a different row.

[Format]

B	C	D
ShiftRow	*SheetCode*	*Shift*

[Arguments]

Argument	Description
SheetCode	Sheet code for output
Shift	Number of rows to shift from the current row

5.5.2 Standard output

ATPL includes an output function called standard output. Worksheets that are output using the standard output function are called "standard output sheets." Standard output sheets are used without having to open any worksheets with the OpenWs statement. The worksheet name is set to 'Prn', and the symbol (#) is assigned as the sheet code.

Standard output is mainly used for debug printing. In statements that perform assignment to variables, regression calculation, or interpolation, specifying (#) as an argument in column F causes the assignment results to be sent to the standard output. Scatter diagrams obtained by Plot statements and regression results (correction terms) obtained by Regress statements are also sent to the standard output sheet.

5.5.3 Input statements

- **ImpCns statement**

The ImpCns statement is used to read data from a **Parameter Type Data Table** in a worksheet, and then store the data as parameters. If the specified parameters have not yet been declared, a declaration of parameters is also performed.

[Format]

B	C	D	E	F
ImpCns	!Table	(*TableName*)	*SheetCode*	
	(!Def)	*Default*		
	(!Col)	*Column*		
	ParName	*ItemName*		
	….	….		

[Arguments]

Argument	Description
TableName	Name of data table in the worksheet. This argument may be omitted. When omitted, the first data table in the specified worksheet is specified.
Default	Value assigned when the specified item is not found. This argument may be omitted. When omitted and the specified item is not found, an error condition will occur.
Column	Column number to be read (1 – 256). This argument may be omitted. Default value is 2.
ParName	Name of the parameter to which data is stored
ItemName	Item name inside the data table
SheetCode	Sheet code of the worksheet that is set to the input device

[Parameter Type Data Table]

Input data needs to be provided in a format called the **Parameter Type Data Table**. This data table can be started from any row number in the worksheet. One data table is considered from the row containing any character in column A to the row before a blank row.

It is possible to define a name to a data table. To define a name, column A of the first row of the data table must contain the word '#Table', and column B must contain the name of the table itself.

The actual data can be written starting from the next row. Item names are written in column A and data in column B.

[Example]

The example program 'ImpCnsTest' specifies table Data2. Therefore, values 150 and 300 are stored in parameters A and B, respectively. By changing the table name in the third row to Data1 or omitting the !Table label in this example, table Data1 is specified instead of Data2, resulting in values 100 and 200 being stored in parameters A and B.

Example program ImpCnsTest

	A	B	C	D	E
1		Program	ImpCnsTest		
2		OpenWS	!Import	DAT	<INP
3		ImpSrs	!Table	Data2	<INP
4			A	PAM1	
5			B	PAM2	
6		End			

Contents of input data sheet 'DAT'

	A	B	C
	#Table	DATA1	
	PAM1	100	
	PAM2	200	
	PAM3	400	
	#Table	DATA2	
	PAM1	150	
	PAM2	300	
	PAM3	600	

- **ImpSrs statement**

The ImpSrs statement is used to read data from a **Time Series Type Data Table** in a worksheet, and then store the data as time-series variables. If the specified time-series variables have not yet been declared, a declaration of time-series variables is also performed.

[Format]

B	C	D	E	F
ImpSrs	!Table	(*TableName*)	*SheetCode*	
	(!Key)	*KeyField*; *KeyField*; ...		
	(!Def)	*Default*		
	SerName	*ItemName; ItemName; ...*		
	....	....		

[Arguments]

Argument	Description
TableName	Name of data table in the worksheet. This argument may be omitted. When omitted, the first data table in the specified worksheet is specified.
KeyField	Name of key field. When specifying more than one, use semicolon (;) as the delimiter.
Default	Value assigned when the specified item is not found. This argument may be omitted. If omitted and the specified item is not found, an error condition will occur.
SerName	Name of the time-series variable to which data is stored
ItemName	Name of the item to be imported. When specifying more than one, use semicolon (;) as the delimiter. The number of *ItemName* and the number of *KeyField* should be matched.
SheetCode	Sheet code of the worksheet that is set to the input device

[Time Series Type Data Table]

Input data needs to be provided in a format called the **Time Series Type Data Table**. This data table can be started from any row number in the worksheet. One data table is considered from the row containing any character in column A to the row before a blank row.

It is possible to define a name to a data table. To define a name, column A of the first row of the data table must contain the word '#TABLE', and column B must contain the name of the table itself.

Column A of the next row needs to contain a character string (usually the time-series name is used), and column B needs to contain a time element. The actual data can be written starting from the next row. Item names must be written in column A, and data needs to be written in column B as well, with each data item written in the position corresponding to its time element.

[Example]

The example program 'ImpSrsTest' first declares three time elements, the years 1990, 1995, and 2000, by the Series statement, and then stores values POP_A and GDP_A contained in the table 'DATA1' in the worksheet 'DAT' into time-series variables POP and GDP, respectively.

Example program ImpSrsTest

	A	B	C	D	E
		Program	ImpSrsTest		
		OpenWS	!Import	DAT	<1
		Series	!Import	YEAR	<1
			!Table	DATA1	
		ImpSrs	!Table	Data	<1

			POP	POP_A	
			GDP	GDP_A	
		End			

Contents of Input data sheet 'DAT'

	A	B	C	D	E
	#TABLE	DATA1			
	YEAR	1990	1995	2000	
	POP_A	1,000	1,010	1,020	
	GDP_A	1,000	1,050	1,120	

- **ImpElm statement**

The ImpElm statement is used to read data from a **Parameter Type Data Table with Row Items** in a worksheet, and then store them as parameters. If specified parameters have not yet been declared, then the declaration of parameters is also performed. The ImpElm statement is different from the ImpCns statement in that it specifies the names of column items.

[Format]

B	C	D	E	F
ImpElm	!Table	(*TableName*)	*SheetCode*	
	(!Def)	*Default*		
	ParName	*RowItemName*,*ColItemName*		
	....	....		

[Arguments]

Argument	Description
TableName	Name of data table in the worksheet. This argument may be omitted. If omitted, the first data table in the specified worksheet is specified.
Default	Value assigned when the specified item is not found. This argument may be omitted. When omitted and the specified item is not found, an error condition will occur.
ParName	Name of parameter to which data is stored.
RowItemName	Name of row item in the data table.
ColItemName	Name of column item in the data table.
SheetCode	Sheet code of the worksheet that is set to the input device.

[Parameter Type Data Table with Row Items]

Input data needs to be provided in a format called **Parameter Type Data Table with Row Items**. The data table can be started from any row number in the worksheet. One data table is considered from the row containing any character in column A to the row before a blank row.

In the first row in the data table, the names of column items must be provided. In this row, a character string (whatever character string is allowed) must be filled in column A, with column item names filled in columns B and onwards.

The actual data can be written starting from the next row. Row item names must be filled in column A, and the data for each row item must be written in columns B and onwards of the corresponding row.

It is possible to assign a name to a data table. To define a name, column A of the first row of the data table (the row above the row containing the column item names) must contain the word "#TABLE", and column B must contain the name of the table.

[Example]

In the following example, numerical value 11 is output to cell [1,2] in worksheet 'sheet1'.

	A	B	C	D	E
		Program			
		OpenWs	!Import	DAT	<1
		ImpElm	!Table	Data1	<1
			POP_95	POP_A,1995	
		end			

Contents of Input data sheet 'DAT'

	A	B	C	D	E
	#TABLE	DATA1			
	YEAR	1990	1995	2000	
	POP_A	1,000	1,010	1,020	
	GDP_A	1,000	1,050	1,120	

- **Read statement**

The Read statement is used to read the value in a worksheet cell.

[Format]

B	C	D	E
Read	*ParName*	*RowNum, ColNum*	*SheetName*

[Arguments]

Argument	Description
ParName	Name of the parameter to which data is stored
RowNum	Column number of cell (1 – 256)
ColName	Row number of cell (1 – 65536)
SheetName	Name of worksheet

[Example]

In the following example, 3 is assigned to the parameter Pam.

	A	B	C	D	E
		Program			
		Read	Pam	1,3	Dat
		end			

Contents of the worksheet 'Dat'

	A	B	C	D	E	F
	1	2	3	4	5	6
	7	8	9	10	11	12

5.5.4 Output statements

- **ExpCns statement**

The ExpCns statement is used to output data stored in variables as Parameter Type Data Tables in the worksheet.

[Format]

B	C	D	E
ExpCns	!Table	(*TableName*)	*SheetCode*
	ItemName	*Expression* (;*Expression*;...)	
	...	...	

[Arguments]

Argument	Description
TableName	Name of data table. This argument may be omitted. When omitted, definition row of the data table name is not output. For more on the output-starting row, refer to [output starting row] below.
ItemName	Item name output in column A of the data table.
Expression	Expression of output data. When delimited by a semicolon (;), the calculation results of the first expression are output to column B, and the calculation results of the next expression are output to column C, and so on.
SheetCode	Sheet code of worksheet to which data table is output.

[Output starting row]

Value of NROW	*TableName*	Output starting row
1	Not to be omitted	1
	Omitted	1
Other than 1	Not to be omitted	NROW+ 1
	Omitted	NROW

NROW: Value of output row counter before execution of instruction statement

[DO-type sequence specification]

An output specification called **DO-type Sequence Specification** can be used in the *Expression* argument. The DO-type specification has to be used with indexes. When members of $FOS are COL, OIL, and GAS, for example, it can be used as follows:

Program code	As interpreter treats it
DO($FOS)	"COL";"OIL";"GAS"
DO($FOS,A[$FOS])	A[COL];A[OIL];A[GAS]

- **ExpSrs and ExpSrs+ statements**

These statements are used to output data stored in variables as time-series type data tables in a worksheet. The ExpSrs statement outputs time-variable (ex. 'year') in the first row. On the other hand, the ExpSrs+ statement does not output this.

[Format]

B	C	D	E	F
ExpSrs	!Table	(*TableName*)	*SheetCode*	*SheetCode*
	ItemName	<*Expression*>	(*Description*)	(*Description*)
	...	...	...	...

[Arguments]

Argument	Description
TableName	Name of data table. This argument may be omitted. When omitted, the definition row of the data table name is not output. For more on output starting row, see [Output starting row] below.
ItemName	Name of item.
Expression	Expression of output data. Time element containing mathematical operation that has a blank (such operations essentially constitute errors) is output as blank.
Description	This argument may be omitted. Writing a character string here causes the specified character string to be output after the last time element.

[Output starting row]

Value of NROW	*TableName*	Output starting row
1	Not to be omitted	1
	Omitted	1
Other than 1	Not to be omitted	NROW+ 1
	Omitted	NROW

NROW: Value of output row counter before execution of instruction statement

- **RwtCns statement**

The RwtCns statement is used to change the data item specified by the *ItemName* argument of the Parameter Type Data Table, specified with the *TableName* argument in the worksheet, to the value of the parameter specified by the *ParName* argument.

[Format]

B	C	D	E	F
RwtCns	!Table	(*TableName*)	*SheetName*	
	(!Col)	*Column*		
	ItemName	<*Expression*>(; <*Expression*>; ...)		
	...	...		

[Arguments]

Argument	Description
SheetName	Name of worksheet in which the data table exists
TableName	Name of data table
ItemName	Name of item
Expression	Expression of output data. When delimited by a semicolon (;), the operation result of the first expression is output to the column that the column number is argument *column*, and operation result of the next expression is output to column that the column number is argument *column* + 1, and so on.
Column	Column number of worksheet to be read out data (1 – 256). This argument may be omitted. Default value is 2.

- **RwtSrs statement**

The RwtSrs statement changes the data item specified by the *ItemName* argument of the Time Series Type Data Table, specified by the *TableName* argument in the worksheet, to the value of the time-series variable specified by the *SerName* argument.

[Format]

B	C	D	E	F
RwtSrs	!Table	(*TableName*)	*SheetName*	
	ItemName	*SerName*		
	...	...		

[Arguments]

Argument	Description
SheetName	Name of worksheet in which the data table exists
TableName	Name of data table
ItemName	Name of item
SerName	Name of time-series variable

- **Write statement**

The Write statement is used to output data to the specified worksheet cell.

[Format]

B	C	D	E
Write	*RowNum*, *ColNum*	*Expression*	*SheetName*

[Arguments]

Argument	Description
RowNum	Column number of cell (1 – 256)
ColName	Row number of cell (1 – 65536)
Expression	Expression of output data
SheetName	Name of worksheet

[Example]

In the following example, numerical value 11 is output to cell [1,2] in the worksheet 'sheet1'.

B	C	D	E
Program			
CalCns	X	:10	
Write	1,2	X+1	sheet1
end			

- **Print and Print+ statement**

These statements are used to output data to the standard output sheet. Print+ statement changes the color of output line into gray.

[Format]

B	C	D
Print	*Expression*	*Expression*

[Arguments]

Argument	Description
Expression	Expression of the output data. The operation result of expression in column C is output to column A on the standard output sheet, and the operation result of expression in column D is output to column B.

- **Plot statement**

The Plot statement is used to create a scatter diagram to the standard output sheet. The data specified for the X and Y-axes are also output to the standard output sheet.

[Format]

B	C	D	E
Plot	(!Style)	*PlotStyle*	
	(!Ctitle)	*ChartTitle*	
	(!Xtitle)	*X-axisTitle*	
	(!Ytitle)	*Y-axisTitle*	
	(!Regend)	*RegendFlg*	
	X-item	*Expression-x*	
	Y-item	*Expression-y*	
	...	...	

[Arguments]

Argument	Description
PlotStyle	Plot style. This argument may be omitted. Default value is 1. 1: Row style 2: Marker style 3: Only the time-series specified latest is row style, and the rest is maker style.
ChartTitle	Title of the plot. This argument may be omitted. If omitted, the title is not output.
X-axisTitle	Title of X axis. This argument may be omitted. If omitted, the title is not output.
Y-axisTitle	Title of Y axis. This argument may be omitted. If omitted, the title is not output.
RegendFlg	Controls output of the legend either ON/OFF. This argument may be omitted. Default is ON.
x-item	Name of output item on the X axis
y-item	Name of output item on the Y axis
Expression-x	Expression of the X-axis data. A time element containing a mathematical operation that has a blank (such an operation essentially constitutes an error) is output as blank.
Expression-y	Expression of the Y-axis data. A time element containing mathematical operation that has a blank (such an operation essentially constitutes an error) is output as blank.

5.5.5 Handling database statements

- **Filter statement**

The Filter statement is used to extract data from a Microsoft Access file, and output to a worksheet.

[Format]

B	C	D
Filter	!File	<*FileName*>
	!Table	*TableName*
	!Sheet	*SheetName*
	KeyfField; *KeyField*; ...	*Keyword*; *Keyword*; ...
	...	...

[Arguments]

Argument	Description
FileName	Name of the database (Microsoft Access file) as full path. When the symbols '.\' are used as the first two characters, path specification starts from the directory where the active workbook is located.
Table	Table name of the database
SheetName	Name of the worksheet to which data is outputted.
KeyField	Name of key field. When specifying more than one, use semicolon (;) as the delimiter.
Keyword	Keyword. Wild cards (*) and (?) can be used. When specifying more than one, use semicolon (;) as the delimiter. The number of arguments *Keyword* should match to the number of arguments *KeyField*.

[Example]

B	C	D
PROGRAM		
FILTER	!file	".\myfile.mdb"
	!table	pop
	!sheet	wrk
	ITEM;country	POP_*;JPN
END		

- **GrpCns statement**

The GrpCns statement is used to read data from a Microsoft Access file, and then store the data as parameters. If the specified variables have not yet been declared, a declaration of variables is also performed.

[Format]

B	C	D	E
GrpCns	!File	<*FileName*>	
	!Table	*TableName*	
	(!Value)	*ValueField*	
	!Key	*KeyField*; *KeyField*; ...	
	(!Def)	*Default*	
	ParName	*ItemName*; *ItemName*; ...	
	...	...	

[Arguments]

Argument	Description
FileName	Name of the database (Microsoft Access file) as full path. When the symbols '.\' are used as the first two characters, path specification starts from the directory where the active workbook is located.
TableName	Table name of the database
ValueField	Name of value field. This argument may be omitted. Default value is 'Value'
KeyField	Name of key field. When specifying more than one, use semicolon (;) as the delimiter.
Default	Value assigned when specified item is not found. This argument may be omitted. If omitted and the specified item is not found in the table, an error condition will occur.
ItemName	Name of the item to be extracted. When specifying more than one, use semicolon (;) as the delimiter. The number of arguments *ItemName* should match to the number of arguments *KeyField*.
ParName	Name of the parameter to which data is stored.

[Example]

	B	C	D	E	F
	PROGRAM				
	GRPCNS	!FILE	".\myfile.mdb"		
		!TABLE	ELS		
		!VALUE	VALUE		
		!KEY	COUNTRY;SECTOR		
		ELS_INDS	JPN; INDS		
		ELS_TRNS	JPN; TRNS		

	END				

The Field structure of Table 'ElS' in file 'myfile.mdb'

Field Name	Description
COUNTRY	Code of country
SECTOR	Code of sector
VALUE	Value of elasticity

- **GrpSrsCrs statement**

The GrpSrsCrs statement is used to read data from a Microsoft Access file, and then store the data as time-series variables. If the specified variables have not yet been declared, a declaration of variables is also performed.

[Format]

B	C	D	E	F
GrpSrsCrs	!File	*<FileName>*		
	!Table	*TableName*		
	!Key	*KeyField*; *KeyField*; ...		
	(!Def)	*Default*		
	SerName	*ItemName* ; *ItemName*; ...		
	...	...		

[Arguments]

Argument	Description
FileName	Name of the database (Microsoft Access file) as full path. When the symbols '.\' are used as the first two characters, path specification starts from the directory where the active workbook is located.
TableName	Table name of the database
KeyField	Name of key field. When specifying more than one, use semicolon (;) as the delimiter.
Default	Value assigned when the specified item is not found. This argument may be omitted. If omitted and the specified item is not found, an error condition will occur.
ItemName	Name of the item to be extracted. When specifying more than one, use semicolon (;) as the delimiter. The number of *ItemName* and the number of *KeyField* should be matched.
SerName	Name of the time-series variable to which data is stored

[Example]

	B	C	D
	PROGRAM		
	SERIES	YEAR	!DIR;1990,1995,2000
	GrpSrsCrs	!file	".\myfile.mdb"
		!table	TRNS
		!key	COUTRY;ITEM
		!def	""
		DRV[TRN]	JPN;CAR
	END		

Field Structure of Table 'TRNS' in file 'myfile.mdb'

Field Name	Data Type	Description
COUNTRY	Text	Code of country
ITEM	Text	Code of item
REFERENCE	Text	Source of data
1980	Number	Value in 1980
1981	Number	Value in 1981
...	...	...
2002	Number	Value in 2002

- **GrpSrsNml statement**

The GrpSrsNml statement is used to read data from a Microsoft Access file, and then store the data as time-series variables. If the specified variables have not yet been declared, a declaration of variables is also performed.

[Format]

B	C	D	E	F
GrpSrsNml	!File	<*FileName*>		
	!Table	*TableName*		
	(!Time)	*TimeField*		
	(!Value)	*ValueField*		
	!Key	*KeyField*; *KeyField*; ...		
	(!Def)	*Default*		
	SerName	*ItemName*; *ItemName*; ...		
	...	...	...	...

[Arguments]

Argument	Description
FileName	Name of the database (Microsoft Access file) as full path. When the symbols '.\' are used as the first two characters, path specification starts from the directory where the active workbook is located.
TableName	Table name of the database
TimeField	Name of time field. This argument may be omitted. Default value is 'Year'.
ValueField	Name of value field. This argument may be omitted. Default value is 'Value'
KeyField	Name of key field. When specifying more than one, use semicolon (;) as the delimiter.
Default	Value assigned when the specified item is not found. This argument may be omitted. If omitted and the specified item is not found, an error condition will occur.
ItemName	Name of the item to be extracted. When specifying more than one, use semicolon (;) as the delimiter. The number of *ItemName* and the number of *KeyField* should be matched.
SerName	Name of the time-series variable to which data is stored

[Example]

	B	C	D
	PROGRAM		
	SERIES	YEAR	!DIR;1990,1995,2000
	GrpSrsNml	!File	"myfile.mdb"
		!Table	TRNS
		!Time	Year
		!Value	Value
		!Key	COUNTRY;ITEM
		DRV[TRN]	JPN; CAR
	END		

Field Structure of Table 'TRNS' in file 'MYDB.mdb'

Field Name	Data Type	Description
Country	Text	Code of country
Item	Text	Code of item
Year	Number	Year
Value	Number	Value

5.6. Assignment statements

- **CalCns statement**

The CalCns statement is used to assign the calculation results of the *Expression* argument to the parameter specified by the *ParName* argument.

[Format]

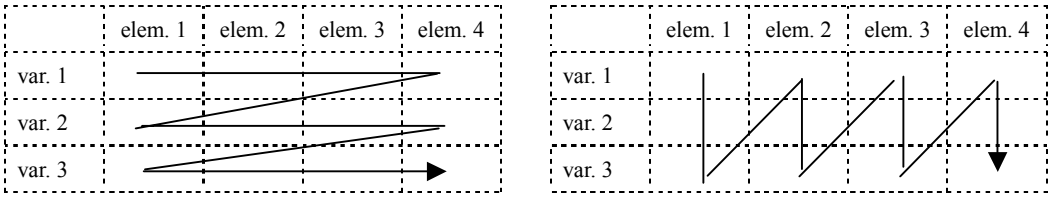
B	C	D	E	F
CalCns	*ParName*	<*Expression*>		(*StnCode*)
	…	…		

[Arguments]

Argument	Description
ParName	Name of parameter corresponding to the left side of the assignment expression.
Expression	Expression corresponding to the right side of the assignment expression.
StnCode	When '#' is specified, the data stored in variable is dumped to the standard output.

- **CalSrs and CalSrsV statements**

These statements are used to assign the operation results of expressions to the time-series variables. In CalSrs and CanSrsV, the calculation order is different. The difference is shown in the following figures.



CalSrs statement CalSrsV statement

Fig. Difference of the calculation order in CalSrs and CalSrsV statements

[Format]

B	C	D	E	F
CalSrs	*SerName*	<*Expression*> (;<*Expression*> (;<*Expression*>))		(*StnCode*)
	…	…		...

[Arguments]

Argument	Description
SerName	Name of time-series variable corresponding to the left side of the assignment expression.
Expression	- Expression corresponding to the right side of the assignment expression. - If a mathematical operation containing a blank is performed, an error condition occurs. However, if there is a ampersand (&) after the expression, an error does not occur and blank is assigned to the time element. - When two expressions are specified by using semicolon delimiter, first expression is used for first time element, and second expression is used for other time elements. - When three expressions are specified by using semicolon delimiter, the first expression is used for the first element, and the third element is used for the last element, and the second expression is used for other elements. - If the expression is the same as the expression immediately before that, tilde may be specified instead of specifying the same expression twice.
StnCode	When '#' is specified, the data stored in variable is dumped to the standard output.

- **CalElm statement**

The CalElm statement is used to assign the operation results of the expression to the time element.

[Format]

B	C	D	E	F
CalElm	*SerName*{*TimeName*}	<*Expression*>		(*StnCode*)
	…	…		...

[Arguments]

Argument	Description
SerName	Name of time-series variable
TimeName	Name of time element
Expression	Expression corresponding to the right side of the assignment expression.
StnCode	When '#' is specified, the data stored in variable is dumped to the standard output.

[Example]

In the following example, 1000 is assigned to the time element of 1990 of the time-series variable POP.

B	C	D	E	F
CalElm	POP{1990}	:1000		

➢ **Count statement**

The Count statement looks at each element of the time-series variable, and then counts the number of non-blank elements and assigns the resulting count to the parameter.

[Format]

B	C	D	E	F
Count	*ParName*	*SerName*		(*StnCode*)

[Arguments]

Argument	Description
ParName	Name of parameter
SerName	Name of time-series variable
StnCode	When '#' is specified, the data stored in variable is dumped to the standard output.

[Example]

In the following example, the time elements of the time-series variable POP are examined, and the number of non-blank elements are counted and assigned to Num.

B	D	E	F
Count	Num	POP	

➢ **CountIf statement**

The CountIf statement looks at each element of the time-series variable, and then counts the number of elements that satisfies the criteria, and assigns the resulting count to the parameter.

[Format]

B	C	D	E	F
CountIf	*ParName*	*SerName, Condition*		(*StnCode*)

[Arguments]

Argument	Description
ParName	Name of parameter
SerName	Name of time-series variable
Condition	Condition expression (for example, =0)
StnCode	When '#' is specified, the data stored in variable is dumped to the standard output.

[Example]

In the following example, time elements of the time-series variable POP are examined, and the number of elements that are greater than 100 are counted and assigned to Num.

B	D	E	F
CountIf	Num	POP,>100	

- **Search statement**

The Search statement looks at elements of the time-series variable from either the beginning or the end, and then assigns the first non-blank element found to the parameter.

[Format]

B	C	D	E	F
Search	*ParName*	*SerName, Direction*		(*StnCode*)

[Arguments]

Argument	Description
ParName	Name of parameter
SerName	Name of time-series variable
Direction	Direction of search !F: From the beginning to the end. !L: From the end to the beginning.
StnCode	When '#' is specified, the data stored in variable is dumped to the standard output.

[Example]

In the following example, the element name of the first time element that exceeds 100 in the time-series variable POP is searched for, and then assigned to the parameter FstYr.

B	C	D
CalSrs	FlgPop	if(POP < 100,"",1)
Search	FstYr	FlgPop,!F

➢ **WsFunc statement**

The WsFunc statement is used to use worksheet function.

[Format]

B	C	D	E	F
WsFunc	*Parname*	*Wsfunction* (*SerName*)		(S*tnCode*)
	...	...		...

[Arguments]

Argument	Description
Parname	Name of parameter to which the calculation result is stored.
Wsfunction	Name of worksheet function
Sername	Name of time-series variable.
Stncode	When (#) is specified, the data stored in variable is dumped to the standard output.

5.7. Regression calculation and interpolation

➢ **Regress statement**

The Regress statement applies least-squares method to dependent variables and independent variables to find the best fit linear function, and then stores in memory the coefficient and Y-axis intercept (constant) that describe this linear function, as well as the contribution ratio indicating the degree of fitness of the linear function. (Coefficient, constant, and contribution ratios are also referred to as "regression results.") The regression results are stored in memory until the next regression calculation is performed or the program is terminated. Regression results stored in memory may be assigned to a parameter by using the GetCef statement. Also, a correction term related to the regression row is sent to the standard output.

The Linear Function Equation can also be expressed as follows:

y = mx + b or y = m1*x1 + m2*x2 + ... + b (if more than one independent variables specified)

Here, the dependent variable y is a function of the independent variable x. 'm' is a coefficient corresponding to each x value, and 'b' is a constant called the "y intercept." In a regression calculation, the LINEST function in Excel is used. Details on correction terms and more can be found in the online help for the LINEST function.

[Format]

B	C	D
Regress	(!Print)	*PrintFlag*
	(!Const)	*ConstantName*
	(!Origin)	*OriginFlag*
	!Y	*<DependentVariable>*
	!X	*<IndependentVariable>*
	...	...

[Arguments]

Argument	Description
PrintFlag	Specifies the standard output. When this argument is set to ON, the correction terms are output to the standard output sheet. When set to OFF, they are not output. This argument may be omitted. Default is ON.
ConstantName	Specifies the name of the regression coefficient. This argument may be omitted. Default value is 'C'.
OriginFlag	When this argument is set to False, the value of constant b becomes 0. This argument may be omitted. Default is True.
DependentVariable	Expressions that are treated as dependent variable. Specified by constants, parameters, time-series variables, time-series variables with the suffix, and expressions containing them.
IndependentVariable	Expressions that are treated as independent variables. Specified by constants, parameters, time-series variables, time-series variables with the suffix, and expressions containing them.

Contents of correction term output

A	B	C	D
R2,Sey	*r2*	*sey*	
F, df	*F*	*df*	
C[0]	*b*	se_b	t_b
C[1]	cef_1	se_1	t_1
...	...	...	...
C[*n*]	cef_n	se_n	t_n
C[e]	*err*		

Description of correction terms

Correction term	Description
R2	Contribution ratio
sey	Standard error from estimated y value.
F	F test or a value that can be regarded as F test.
df	Degree of freedom.
se_0	Value of standard error from constant b. (When *OriginFlag* of Regres statement is FALSE, se_b = #N/A .)
$se_1, se_2, ..., se_n$	Values of standard error from coefficients $cef_1, cef_2, ..., cef_n$.
$t_b, t_1, t_2, ..., t_n$	t-test values for constants and coefficients.

[Example]

B	C	D
Regress	!Const	C
	!Print	On
	!y	ln(ENE)
	!x	ln(GDP)

- **GetCef statement**

The GetCef statement is used to assign to parameters the coefficients, constants, and contribution ratios (known together as "regression results") obtained from regression calculation. The calculation results of the *Expression* argument are assigned to the parameter specified by the *ParName* argument. Regression results are stored in memory until the next regression calculation is performed or the program is terminated.

[Format]

B	C	D	E	F
GetCef	*ParName*	<*Expression*>		[*StnCode*]
	...	...		...

[Arguments]

Argument	Description
ParName	Name of parameter corresponding to the left side of the assignment expression.
Expression	Expression corresponding to the right side of the assignment expression.
StnCode	When (#) is specified, sends calculation results to the standard output.

➢ **InterPol statement**

The InterPol statement looks at time elements of the time-series variable specified by the *SerName* argument, and assigns to those elements that have blank values, the values obtained from linear interpolation. When the first or last time element is blank, the value of the closest time element that has data is used for that element.

[Format]

B	C	D	E	F
InterPol	*SerName*			*StnCode*
	...			...

[Arguments]

Argument	Description
SerName	Name of time-series variable to be interpolated.
StnCode	When (#) is specified, sends interpolation results to the standard output.

5.8. Control statements

➢ **If / ElseIf / Else / EndIf statement**

Based on the status of the program, the If statement branches the process. An If statement cannot be nested inside another If statement.

[Format]

B	C	D	E	F
If	*Condition*			
(Block 1)				
ElseIf	*Condition*			
(Block 2)				
ElseIf	*Condition*			
(Block 3)				
Else				
(Block n)				
EndIf				

[Argument]

Argument	Description
Condition	Logical conditional expression

- **Goto and Label statements**

These statements cause the process to jump to the specified label unconditionally.

[Format]

B	C	D	E	F
Goto	*GotoLabel*			
(Statements)				
Label	*LabelName*			

[Arguments]

Argument	Description
GotoName	Name of label to jump to
LabelName	Name of label

- **For / Next statements**

All statements between the For statement and Next statement are repeated. Members of the index specified by the *LoopIndex* argument are called the loop members. The number of repetitions will be the number of loop members. As the repetition process continues, loop members are assigned to the *ParName* argument one by one. By specifying $Srs as the *LoopIndex*, time element declared with the Series statement can be used as a loop member. A For statement cannot be nested inside another For statement.

[Format]

B	C	D
For	*ParName*	*LoopIndex*
(Statements)		
Next		

- **Stop statement**

The Stop statement terminates execution of the program. When the Stop statement is executed, a message box confirming termination of the program is displayed.

[Format]

B	C
Stop	

5.9. Executing subprograms and VBA macros

➢ **Call statement**

The Call statement is used to execute a subprogram. By specifying arguments with the Call statement, it is possible to pass constants from the main program to the subprogram, and transfer parameters between the main and subprograms. The function of exchanging time-series variables is not provided; the transfer of time-series variables must be performed through worksheets.

[Format] of Call statement

B	C	D
Call	*ProgramName*	(*Arg* ; *Arg*; ...)

[Arguments] of Call statement

Argument	Description
ProgramName	Name of the subprogram to be executed. The subprogram must be written in the same worksheet.
Arg	Argument of the Call statement. Constants and parameters can be used. When specifying more than one, use a semicolon (;) as the delimiter. When parameters are used as arguments, values will be returned after execution of the subprogram.

[Format] of subprogram

B	C	D
Program	*AreaName*	(*Arg* ; *Arg*; ...)
	&	(*Arg* ; *Arg*; ...)
(Statements)		
End		

[Example]

An example of using the Call statement is provided below. Executing the following program will display "20" in the upper part of a message box, and "30" in the lower part.

	A	B	C	D	E
		Program	Main		Main Area
		CalCns	Arg1	:20	
			Arg2	:0	
		Call	Test	10;Arg1;Arg2	
		Message	Arg1	Arg2	
		End			

		Program	Test	A;B;C	Sub Area
		CalCns	C	A+B	
		End			

- **VBA statement**

The VBA statement is used to execute a VBA macro.

[Format]

B	C
VBA	*MacroName*

[Arguments]

Argument	Description
MacroName	Name of the VBA macro. Only macros that are written in the same workbook can be executed.

5.10. Handling files and directories

- **MkDir statement**

The Mkdir statement is used to make a directory.

[Formant]

B	C
MkDir	*<DirectorynName>*

[Argument]

Argument	Description
DirectoryName	This argument specifies the directory name by using an expression. When the symbols (.\) symbols are used as the first two characters, path specification starts from the directory where the workbook is located.

[Example]

B	C
MkDir	".\DATA"

➢ **Load and Load- statements**

These statements load the contents of a text file into a worksheet. Tab symbols are recognized as delimiters. If the file specified by the Load statement does not exist, an error message is displayed and the process is canceled. If the file specified by the Load- statement does not exist, nothing happens and the process proceeds to the next instruction statement.

[Format]

B	C	D
Load	*SheetName*	*<FileName>*

[Arguments]

Argument	Description
SheetName	Name of the worksheet into which data is read. If the specified worksheet does not exist, it will be added automatically.
FileName	This argument specifies the file name as a full path by using an expression. When the symbols (.\) symbols are used as the first two characters, path specification starts from the directory where the workbook is located.

[Example]

B	C	D
Load	Dat	".\UserDataA\JPN\JPN-ENG.DAT"

➢ **Save statement**

The Save statement is used to save data in a worksheet as a text file (tab-delimited file).

[Format]

B	C	D
Save	*SheetName*	*<FileName>*

[Arguments]

Argument	Description
SheetName	Name of the worksheet in which data to be saved is located. If the specified worksheet does not exist, an error condition will occur.
FileName	This argument specifies the file name as a full path by using an expression. When the symbols (.\) are used as the first two characters, path specification starts from the directory where the workbook is located.

[Example]

B	C	D
Save	Dat	".\UserDataA\JPN\JPN-ENG.DAT"

5.11. Handling worksheets

- **DelWs statement**

The DelWs statement is used to delete a worksheet.

[Format]

B	C
DelWs	*SheetName*

[Arguments]

Argument	Description
SheetName	Name of the worksheet to be deleted. If the specified worksheet does not exist, nothing will be done.

[Example]

A	C
DelWs	Dat

- **AddWs statement**

The AddWs statement is used to add a worksheet.

[Format]

B	C
AddWs	*SheetName*

[Arguments]

Argument	Description
SheetName	Name of the worksheet to be added. If the specified worksheet already exists, nothing will be done.

[Example]

B	C
AddWs	Dat

- **RenameWs statement**

The RenameWs statement is used to rename a worksheet.

[Format]

B	C	D
RenameWs	*OldName*	*NewName*

[Arguments]

Argument	Description
OldName	Name of the worksheet to be renamed. If the specified worksheet does not exist, an error condition will occur.
NewName	Name of the worksheet after renaming

[Example]

B	C	D
RenameWs	Dat1	Dat2

- **PrintWs statement**

The PrintWs statement is used to print out a worksheet.

[Format]

B	C
PrintWs	*SheetName*

[Arguments]

Argument	Description
SheetName	Name of the worksheet to be printed. If the specified worksheet does not exist, an error condition will occur.

[Example]

B	C
PrintWs	Dat

- **ClearWs statement**

The ClearWs statement is used to clear the contents of a worksheet.

[Format]

B	C
ClearWs	*SheetName*

[Arguments]

Argument	Description
SheetName	Name of the worksheet whose contents are to be cleared. If the specified worksheet does not exist, an error condition will occur.

[Example]

B	C
ClearWs	Dat

- **CopyWs statement**

The CopyWs statement is used to copy the worksheet into the specified workbook.

[Format]

B	C	D
CopyWs	*Sheetname*	*Bookname*

[Argument]

Argument	Description
Sheetname	Name of the worksheet which is copied.
Bookname	Name of the workbook into which the worksheet is copied.

[Example]

	B	C	D
	PROGRAM		
	OpenWb+	".\MyBook.xls"	
	CopyWs	MySheet	MyBook.xls
	CloseWb	"MyBook.xls"	
	END		

5.12. Handling workbooks

- **OpenWb and OpenWb+ statement**

These statements are used to open the specified workbook. If the workbook specified by the

OpenWb statement does not exist, an error message is displayed and the process is stopped. If the workbook by the OpenWb+ statemenst does not exist, new workbook is added, and saved as the specified file name in the directory that the workbook under work is located. [Format]

B	C
OpenWb	<*BookName*>

[Argument]

Argument	Description
BookName	The workbook name as a full path by using an expression. When the symbols '.\' are used as the first two characters, path specification starts from the directory where the workbook under work is located.

[Example]

B	C
OpenWb	".\Book1.xls"

- **SaveWb statement**

The SaveWb statement is used to save the specified workbook as a same name.

[Format]

B	C
SaveWb	<*BookName*>

[Argument]

Argument	Description
BookName	The workbook name by using an expression.

[Example]

B	C
SaveWb	"BOOK1.XLS"

- **CloseWb statement**

The CloseWb statement is used to close the specified workbook.

[Format]

B	C
CloseWb	<*Bookname*>

[Argument]

Argument	Description
Bookname	The workbook name by using an expression.

[Example]

B	C
CloseWb	"BOOK1.XLS"

5.13. Other statements

- **TrimSrs statement**

The TrimSrs statement looks at all elements of all declared time-series variables, and then deletes any time elements that are blank. Therefore, execution of this instruction means that the declaration contents of the Series statement will be modified.

For example, here we assume that 1990, 1991, 1992, 1993, and 1994 are declared as time elements, and POP and GDP are declared as time-series variables. We also assume that POP of 1991 and GDP of 1993 are both blank. In this case, executing the TrimSrs statement causes 1991 and 1993 to be deleted from the list of time elements, leaving only 1990, 1992 and 1994 remaining in the list of time elements.

[Format]

B
TrimSrs

- **DelVar statement**

The DelVar statement is used to delete specified variables. As the number of variables increases, execution speed slows down. To maintain optimal execution speed, it is recommended that variables be deleted when they are not needed any longer.

[Format]

B	C
DelVar	*VarName*

[Arguments]

Argument	Description
VarName	Name of variable to be deleted.

- **CopyTable statement**

The CopyTable statement is used to copy a table.

[Format]

B	C	D	E
CopyTable	*TableName*	*ImpCode*	*ExpCode*

[Arguments]

Argument	Description
TableName	Name of table
ImpCode	Sheet code of input worksheet
ExpCode	Sheet code of output worksheet

- **Message statement**

The Message statement is used to display a message box.

[Format]

B	C	D
Message	<*Expression*>	<*Expression*>

[Argument]

Argument	Description
Expression	Character strings to be displayed in the message box. The expressions in column C and D are connected with a carriage return code.

6. Creating User Interfaces

Adding ATML as a button makes it possible to create simple user interfaces. Descriptions here are provided for those who have a basic knowledge of Microsoft's Visual Basic for Applications (VBA).

(1) Creating ATML macros

To create an ATML macro, write 'Macro' in column B and the name of the macro in column C in the same row. Write the macro starting from the row immediately after the first row.

Writing macro

A	B	C	D	E
	Macro	MacroName		
	command	argument	argument	argument
	command	argument	argument	argument

(2) Creating VBA macros

To create a VBA macro, open the Visual Basic Editor and add a standard module into the workbook that is currently in use. Using some valid macro name, add a VBA macro to the standard module and write a program as follows. (In the following example, the name of the VBA macro is Macro1.)

```
Sub Macro1()
  Application.Run "AimTrend.xla!RunATML", MacroSheet, MacroName, [Message],[Continue]
End Sub
```

MacroSheet is the name of the worksheet in which ATML is written, and *MacroName* is the name of macro contained in *MacroSheet*. These arguments must be put in single quotation marks.

Message is an optional argument, whose default value is True. When it is omitted or True is used, a message stating 'Normal end' will be displayed at the end of macro execution. If the value is set to False, this message will not be displayed.

Continue is also an optional argument, and its default value is False. When it is omitted or set to False, the program will stop after macro execution. When its value is set to True, the process proceeds to the next VBA instruction statement, if any.

(3) Creating buttons

To create a button, add a worksheet for the interface, select [View]/[Toolbars]/[Forms] from the Excel menu bar to display the [Forms] toolbar, and then drag and drop a [Button] onto the worksheet.

Then, in the [Add macro] dialog box, select the VBA macro that was created earlier. The button

name can be edited to whatever name you want.

The button on the worksheet is now connected to ATML execution, and clicking the button will cause ATML to execute.

It is also possible to make the interface easier to use by employing forms such as combo, as well as buttons.

7. Index of ATPL Instructions

AddWs....51
CalCns....39
CalElm....40
Call....48
CalSrs and CalSrsV....39
ClearWs....52
CloseWb....54
CloseWs....21
CopyTable....56
CopyWs....53
Count....41
CountIf....41
Data....16
DelVar....55
DelWs....51
Dim....17
ErsIdx....20
ExpCns....28
ExpSrs and ExpSrs+....29
For / Next....47
GetCef....45
Goto....47
GrpSrsCrs....36
GrpSrsNml....37
If / ElseIf / Else / EndIf....46
ImpCns....22
ImpElm....26
ImpSrs....24
Index....18
InterPol....46
Label....47
Load and Load-....50
Message....56
MkDir....49
OpenWb and OpenWb+....53
OpenWs....20
Plot....32
Print and Print+....32
PrintWs....52
Read....27
Regress....43
RenameWs....52
RwtCns....30
RwtSrs....31
Save....50
SaveWb....54
Search....42
Series and Series+....14
ShiftRow....22
Stop....47
TrimSrs....55
VBA....49
Write....31
WsFunc....43